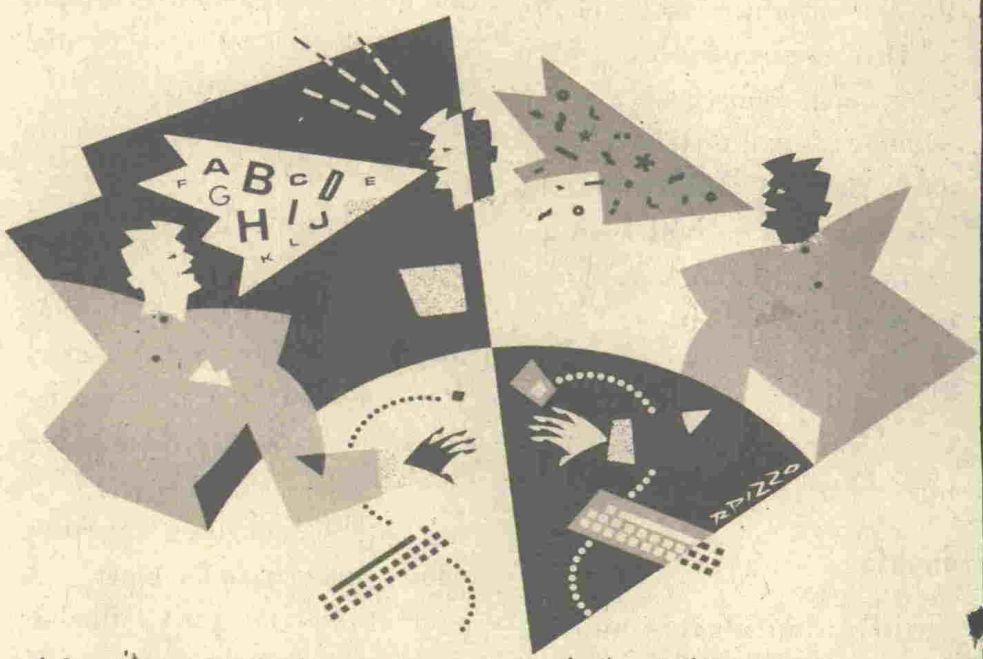
ศึกษาภาษาซีเบื้องต้น

คุณรู้จักภาษาซีมากแค่ไหน ลองศึกษาวิธีการอ่านและเขียนภาษาที่เป็นโครงสร้าง แล้วคุณจะประทับใจในความสามารถ



คนที่เพิ่งเริ่มหัดเขียนภาษาซีใหม่ ๆ อาจจะทำใจลำบากนิดหน่อยในการทำความคุ้นเคยกับสัญลักษณ์ต่าง ๆ ที่ปรากฏในภาษาซีเพราะ ซีเป็นภาษาที่อ่านยากคล้าย ๆ กับรหัสย่ออะไรบางอย่าง ถ้าเปรียบเทียบภาษาซีกับปาสคาลแล้วจะพบว่าเป็นภาษาที่คล้ายกันคือเป็นภาษาโครงสร้าง (structural programming language) อย่างแท้จริง แต่ว่าในขณะที่ปาสคาลเป็นภาษาที่อ่านง่ายมาก คล้ายกับเรียงความภาษาอังกฤษที่เป็นเรื่องเป็นราว ส่วนภาษาซีมีการใช้สัญลักษณ์แทนมาก เช่น ใช้ { (left brace) และ } (right brace) แทนคำว่า BEGIN และ END ในปาสคาล แต่ว่าผู้ที่เคยใช้ปาสคาลมาก่อนแล้วเปลี่ยนมาใช้ซีส่วนมากจะไม่หันกลับไปแตะปาสคาลอีกเลย เพราะซีเป็นภาษาที่มีความยืดหยุ่นและความคล่องตัวสูง ซึ่งสามารถช่วยให้เขียนรหัสได้สั้นกะทัดรัดมาก แต่ส่วนนั้นก็ทำให้รหัสภาษาซีอ่านยาก

บทความนี้จะกล่าวถึงวิธีการอ่านเขียนและการใช้ประโยชน์ของการประกาศตัวแปร (declaration) ในภาษาซี โดยเริ่มจากตัวอย่างง่าย ๆ เช่น int x; คือ x เป็น integer หรือ char ch; คือ ch เป็น character นั่นเอง ประโยชน์ก็คือ x เป็นที่เก็บข้อมูลของตัวเลขฐานสองขนาด 2 หรือ 4 ไบต์ แล้วแต่เครื่องไอบีเอ็มพีซีหรือพวกคอมแพตทิเบิลทั้งหลายจะเป็น 2 ไบต์ แต่ถ้าเป็นมินิคอมพิวเตอร์ก็มักจะเป็น 4 ไบต์ อันนี้แล้วแต่สถาปัตยกรรม (architecture) ของแต่ละเครื่องและมักเก็บเป็นแบบคิดเครื่องหมาย + หรือ - เพราะในซีถ้าจะให้เก็บแบบไม่คิดเครื่องหมายคือมองเป็นบวกอย่างเดียวจะให้ใส่คำว่า unsigned ไว้หน้าเช่น unsigned int u; สำหรับ ch คือ char หรือที่เก็บข้อมูลขนาด 8 บิตหรือ 1 ไบต์ที่มักจะมองเป็นบวกแล้วเราก็เปรียบเทียบเป็นอักขระต่าง ๆ ในภาษามนุษย์ แต่บางทีก็มองเป็น integer ขนาดเล็กที่คิดเครื่องหมายด้วยก็ได้ ทั้งนี้ก็เพื่อประโยชน์ของการจัดเก็บข้อมูลให้ประหยัดหน่วยความจำและดิสก์นั่นเอง

char *x จะอ่านว่า "x เป็นตัวชี้ไปยัง character คือ x จะเก็บตำแหน่ง (address) ของข้อมูลที่เป็น character"

ถ้าเป็น char *x[5]; จะอ่านว่า "x เป็นอะเรย์ของพอยน์เตอร์ 5 ตัวที่ชี้ไปยัง character" (x is an array of five pointers to character)

ส่วน char (*x)[5]; เราต้องอ่านในวงเล็บก่อน กลายเป็น "x เป็นพอยน์เตอร์ที่ชี้ไปยัง character 5 ตัว (x is a pointer to an array of five characters) ซึ่งต่างกับแบบที่แล้ว โดยเปรียบเทียบได้จากรูปที่ 1 และ 2

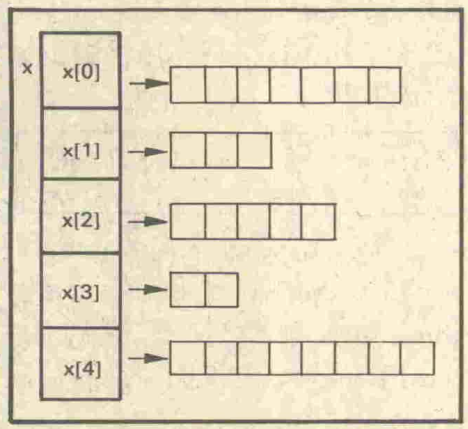
ประโยชน์ของรูปที่ 1 ใช้เก็บ character string หลาย ๆ ชุดหรือข้อความหลาย ๆ ประโยค ส่วนในรูปที่ 2 จะเห็นประโยชน์ชัดเมื่อเปลี่ยน index เป็นเลขจำนวนมาก ๆ หรือเมื่อขนาดไม่

แน่นอน จะใช้โดยการขอ malloc( ) ทำให้ใช้หน่วยความจำได้เต็มที่

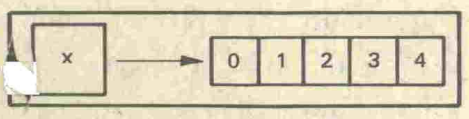
ถ้าเป็นฟังก์ชันอย่าง char *x( ); จะอ่านว่า x เป็นฟังก์ชันที่คืนค่าเป็นตัวชี้ไปยัง character ตัวอย่างเช่น ฟังก์ชันที่จะคืนค่าเป็นชื่อของคนที่มีเลขประจำตัวส่งไปเป็น argument เป็นต้น

สำหรับ char (*x)( ); อ่านว่า x เป็นตัวชี้ไปยังฟังก์ชันที่คืนค่ามาเป็น character ตัวหนึ่ง สมมติเขียนฟังก์ชัน char myfunc( ); ซึ่ง return ค่าเป็น character แทนที่คุณจะใช้มันโดยการเรียก myfunc( ) ตรง ๆ ก็อาจจะเรียกโดยให้ x ไปชี้ก่อนแล้วค่อยใช้ผ่าน x แทน

ถ้าพบ int **(*bin[3] ( ) [5] บ้างอย่าคิดว่าเขียนผิดที่ถ้าคอมพายล์ผ่านและใช้งานได้อ่านว่า "bin เป็นอะเรย์ของตัวชี้ 3 ตัวที่ชี้ไปยังฟังก์ชันซึ่งคืนค่าเป็นอะเรย์ของตัวชี้ 5 ตัวที่ชี้ไปยัง integer" ที่จริงแล้วก็เป็นการผสม 4-5 แบบข้างต้นนั่นเอง สำหรับการใช้งานเมื่อดูจากคำอ่านแล้วก็เป็นไปได้ แต่ก็วุ่นวายพอ ๆ กับการอ่าน คงจะไม่มีใครใช้เพราะเสี่ยงต่อการตกหล่นขั้นใดขั้นหนึ่งทำให้เกิดบักได้ง่าย ๆ



รูปที่ 1 char*x[5]; แสดงชุดของพอยน์เตอร์ที่ชี้ไปยังคาแรกเตอร์



รูปที่ 2 char (*x) [5]; แสดงตัวชี้ไปยังคาแรกเตอร์ 5 ตัว

สัญลักษณ์	คำอ่าน	ลำดับ	ทิศทาง
[ ]	array of	1	Post
( )	function returning	1	Post
*	pointer to	2	Pre
<type>	<type>	3	Pre

ตารางที่ 1 ลำดับและการอ่านสัญลักษณ์ในการประกาศตัวแปร

หลักการอ่าน

การอ่านคำประกาศเหล่านี้มีสูตรตายตัวอยู่ หลักการคือ เรื่องของลำดับก่อนหลัง (priority) ลำดับ 1 จะต้องอ่านก่อนลำดับ 2 ดูจากตารางที่ 1 คอลัมน์ของ "คำอ่าน" บอกว่าเจ้าสัญลักษณ์ในคอลัมน์แรกอ่านอย่างไรในภาษาอังกฤษ คอลัมน์ "ลำดับ" บอกลำดับก่อนหลังในการอ่าน คืออ่าน [ ] (สัญลักษณ์ของอะเรย์) หรือ ( ) (สัญลักษณ์ของฟังก์ชัน) ก่อน * (สัญลักษณ์ของตัวชี้) และก่อนชนิด (หรือ <type>) ตามลำดับ แต่ก็เหมือนกับคณิตศาสตร์ คือถ้ามีวงเล็บก็ต้องอ่านภายในวงเล็บก่อน ส่วนในคอลัมน์ "ทิศทาง" บอกว่าสัญลักษณ์เหล่านี้อยู่ข้างหน้า (pre) (ทางซ้ายของตัวแปร) หรือข้างหลัง (post) (ทางขวาของตัวแปร)

จุดเริ่มต้นเริ่มตรงชื่อของตัวแปร หรือชนิดใหม่ที่เราใช้ typedef ก่อนจะไปทางซ้ายหรือทางขวาก็ดูจากลำดับของสัญลักษณ์ โดยไม่ลืมว่าถ้ามีวงเล็บต้องอ่านในวงเล็บเสียก่อน ในแต่ละชั้นจะมีสัญลักษณ์แบบ post มากกว่า pre ได้

```
TOOL *bin[3];
```

เริ่มจาก "bin is" แล้วจะต่อด้วยสัญลักษณ์อะไรก็ได้ตามตารางที่ 1 เป็น [ ] เพราะ [ ] มีลำดับก่อน * จึงเป็น "bin is array of three pointers to TOOL" หรือ "bin เป็นอะเรย์ของตัวชี้ไปยังข้อมูลชนิด TOOL" ในที่นี้ TOOL จะเป็นอะไรก็แล้วแต่ ไม่ว่าจะเป็น int หรือ char ธรรมดา หรือจะเป็นอะไรก็ได้ เช่น

```
typedef TOOL *BIN__DEF[3];
BIN__DEF bin;
```

จะอ่านว่า "bin is of type BIN__DEF, where BIN__DEF is an array of there elements, each of which is a pointer to TOOL" หรือ "bin เป็นข้อมูลชนิด BIN__DEF, โดย BIN__DEF เป็นชนิดของอะเรย์ของตัวชี้ 3 ตัวซึ่งชี้ไปยังข้อมูลชนิด TOOL"

```
TOOL (*bin) [3];
```

ตัวอย่างนี้มีวงเล็บ ดังนั้นจะต้องพิจารณา * ในวงเล็บก่อน [3] จะได้ว่า "bin is a pointer to an array of three elements of TOOL" หรือว่า "bin เป็นตัวชี้ไปยังอะเรย์ของข้อมูลชนิด TOOL 3 ตัว"

```
typedef TOOL (*BIN__DEF) [3];
BIN__DEF bin;
```

อ่านว่า "bin is of type BIN__DEF, where BIN__DEF is a pointer

to an array of three elements, each of which is of type TOOL" หรือ "bin เป็นข้อมูลชนิด BIN__DEF, โดย BIN__DEF เป็นตัวชี้ไปยังอะเรย์ของข้อมูลชนิด TOOL 3 ตัว"

```
TOOL *bin( );
```

เริ่มจาก "bin is" แล้วจะเป็น * หรือ ( ) ก่อน จากตารางที่ 1 สัญลักษณ์ ( ) (สัญลักษณ์ของฟังก์ชัน) มีลำดับก่อน * (สัญลักษณ์ของตัวชี้) ดังนั้นจะได้ว่า "bin is a function returning a pointer to TOOL" หรือ "bin เป็นฟังก์ชันที่คืนค่าเป็นตัวชี้ไปยังข้อมูลชนิด TOOL"

```
typedef TOOL *BIN__DEF;
BIN__DEF bin( );
```

อ่านว่า "bin is a function of type BIN__DEF, where BIN__DEF is a pointer to TOOL" หรือ "bin เป็นฟังก์ชันชนิด BIN__DEF, โดยที่ BIN__DEF เป็นตัวชี้ไปยังข้อมูลชนิด TOOL" ซึ่งต่างกับ

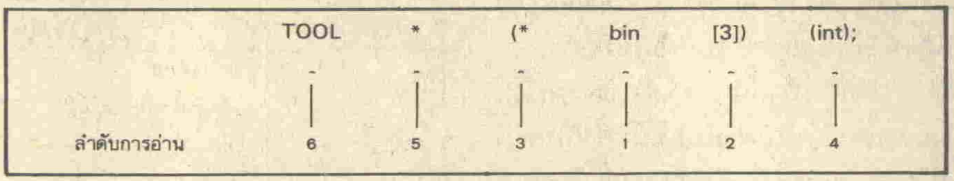
```
TOOL (*bin) ( );
```

อ่านว่า "bin is..." วงเล็บทำให้ *bin มีลำดับก่อน ( ) ดังนั้นเราจะได้ "bin is a pointer to a function returning TOOL" หรือ "bin เป็นตัวชี้ไปยังฟังก์ชันที่คืนค่าเป็นข้อมูลชนิด TOOL"

```
typedef TOOL (*BIN__DEF) ( );
BIN__DEF bin;
```

อ่านว่า "bin is of type BIN__DEF, where BIN__DEF is a pointer to a function returning TOOL" หรือ "bin เป็นข้อมูลชนิด BIN__DEF โดย BIN__DEF เป็นตัวชี้ไปยังฟังก์ชันที่คืนค่าเป็น TOOL"

```
TOOL *bin[3] ( );
```



รูปที่ 3

เริ่มจาก "bin is..." เรารู้ว่า [ ] มีลำดับก่อน * จึงอ่านว่า "bin is an array of three elements of..." ต่อจากนั้นจะเป็น * ก่อน หรือ ( ) ก่อน วิธีการมองคือ เมื่อเราอ่าน [ ] ไปแล้วลองให้ X แทนส่วนของ bin[3] แล้วจะเหลือ TOOL *X( ); จากตารางที่ 1 ทำให้เราเลือก ( ) ก่อน * และจะได้ว่า "X is a function returning a pointer to TOOL"

ดังนั้นทั้งหมดจะได้ว่า "bin is an array of three elements, each of which is a function returning a pointer to TOOL" หรือ "bin เป็นอะเรย์ของฟังก์ชัน 3 ฟังก์ชันที่คืนค่าเป็นตัวชี้ไปยังข้อมูลชนิด TOOL" อย่างนี้ผิด ถึงจะอ่านได้แต่ก็ผิดการใช้ เพราะ อะเรย์สามารถเก็บตัวชี้ไปยังฟังก์ชันได้ แต่ไม่สามารถเก็บตัวฟังก์ชันได้

ค่าที่ถูกต้องคือ TOOL (*bin[3]) ( ) อ่านว่า "bin is an array of three pointers to function returning TOOL" หรือ "bin เป็นอะเรย์ของตัวชี้ไปยังฟังก์ชันที่คืนค่าเป็น TOOL" สังเกตดูว่าจะต้องมีวงเล็บครอบเพื่อจับคู่สัญลักษณ์แบบ post และแบบ pre ในระดับชั้นเดียวกัน

```
typedef TOOL (*BIN__DEF[3]) ( );
BIN__DEF bin;
```

เป็นการใช้ typedef มาร่วมด้วยเหมือนเคย ก็จะอ่านได้ว่า "bin is of type BIN__DEF, where BIN__DEF is an array of three pointers to functions returning TOOL" หรือ "bin เป็นชนิด BIN__DEF, โดย BIN__DEF เป็นอะเรย์ของตัวชี้ 3 ตัว ซึ่งชี้ไปฟังก์ชันที่คืนค่าเป็น TOOL"

```
TOOL *(*bin[3]) (int);
```

int ในวงเล็บนั้นบอกชนิดของ argument ที่จะต้องส่งให้ฟังก์ชัน เมื่ออ่านด้วยกฎเกณฑ์เดิมดังรูปที่ 3 จะได้ "bin is array of three elements, each of which is a pointer to a function whose argument is an integer and which returns a pointer to TOOL" หรือ "bin เป็นอะเรย์ของตัวชี้ไปยังฟังก์ชันที่รับอาร์กิวเมนต์เป็น integer และฟังก์ชันนั้นจะคืนค่าเป็นตัวชี้ไปยัง TOOL"

```
typedef TOOL *(*BIN__DEF[3]) (int);
BIN__DEF bin;
```

ก็เพิ่ม typedef อีกนิดหน่อยเอง อ่านได้ว่า "bin is of type BIN__DEF, where BIN__DEF is an array of three elements, each of which is a pointer to a function whose argument is an integer and which returns a pointer to TOOL" หรือเรียกว่า "bin เป็นอะเรย์ของตัวชี้ 3 ตัวซึ่งชี้ไปที่ฟังก์ชันที่รับค่าอาร์กิวเมนต์เป็น integer และจะคืนค่าเป็นตัวชี้ไปยัง TOOL" การใช้แบบนี้ก็พอจะมีอยู่ คือเรามีฟังก์ชัน

อยู่ชุดหนึ่งที่จะคืนค่าให้เป็น pointer to TOOL และเวลาเรียกจะต้องส่ง integer ไปให้ เราก็เอาตำแหน่งหรือ address ของฟังก์ชันเหล่านั้นมาเก็บไว้ในอะเรย์ เวลาเรียกก็เรียกผ่าน index ของอะเรย์โดยการเปิดตาราง (lookup table) ก็ได้ ตัวตารางที่ว่านี้คือ bin ที่ประกาศยาว ๆ นี้เอง

อ่านแล้วก็ต้องเขียน

สมมติเรามีฟังก์ชันอยู่ MAX__FUNC ฟังก์ชันที่คืนค่าเป็น pointer to char แล้วเราสามารถเรียกใช้มันได้โดยการเปิดตารางจากตัวแปรหนึ่ง สิ่งที่ต้องการคืออะเรย์ชุดหนึ่งที่จะเก็บตำแหน่งหรือ address ของฟังก์ชันพวกนั้น

สมมติอะเรย์นี้ชื่อ bin เริ่มจากคำอ่านในแบบเดิม ๆ จะได้ว่า "bin เป็นอะเรย์ขนาด MAX__FUNC ของตัวชี้ไปยังฟังก์ชันที่คืนค่าเป็นตัวชี้ไปยัง char" หรือ "bin is an array of MAX__FUNC elements of pointers to functions returning pointer to char" เริ่มจากรูปที่ 4 จะได้เป็น

```
char *(*bin[MAX__FUNC]) ( );
```

หรือถ้าเขียนโดยใช้ typedef จะได้

```
typedef char *(*BIN__DEF[MAX__FUNC]) ( );
BIN__DEF bin;
```

วิธีใช้งาน

เมื่ออ่านออกเขียนได้แล้ว ก็ต้องใช้งานให้เป็นถึงจะเกิดประโยชน์ได้ คือจะต้องสามารถเข้าถึง (access) และตั้งค่าให้ข้อมูลเหล่านั้นได้ ตัวอย่างเช่น

```
"bin is"                              จะได้ --> bin
"an array of MAX__FUNC elements"      จะได้ --> bin[MAX__FUNC]
"of pointers"                         จะได้ --> *bin[MAX__FUNC]
"to function"                         จะได้ --> (*bin[MAX__FUNC]) ( )
"returning pointer"                   จะได้ --> *(*bin[MAX__FUNC]) ( )
"to char"                             จะได้ --> char *(*bin[MAX__FUNC]) ( )
```

รูปที่ 4

```
typedef struct {                      /* กำหนดชนิดของ TOOL */
        int field;
} TOOL;
TOOL *bin[3];                         /* กำหนดอะเรย์ขนาด 3 ช่อง */
TOOL a, b, c;                         /* ข้อมูลชนิด TOOL */
bin[0] = &a;                          /* ใส่ address ลงในอะเรย์ */
bin[1] = &b;
bin[2] = &c;
และการใช้ข้อมูล ก็จะเป็นในลักษณะ
int i;                                /* กำหนดตัวแปร integer */
i = bin[0]->field;                    /* วิธีการเข้าถึงข้อมูล */
ตัวอย่างถัดมา ยากขึ้นนิดหน่อย
typedef struct {                      /* กำหนดชนิดของ TOOL */
        int field;
} TOOL;
typedef TOOL                          (*BIN__DEF) [3];
BIN__DEF bin;
```

รูปที่ 5 แสดงตำแหน่งของข้อมูลที่เป็นชนิด TOOL

```
TOOL *bin[3];
```

"bin เป็นอะเรย์ขนาด 3 ตัวของตัวชี้ไปยัง TOOL" เมื่อค่าแต่ละช่องของอะเรย์นี้เป็นตัวชี้ ดังนั้นเราจะต้องหาตำแหน่ง (address) ของข้อมูลที่เป็นชนิด TOOL เข้าไปเก็บ (สมมติ TOOL เป็น struct ที่กำหนดข้างล่างนี้) คือรูปที่ 5

เรารู้ว่า bin เป็นตัวชี้ไปยังอะเรย์ขนาด 3 ช่องของ structure TOOL การให้ค่าจะเป็นในลักษณะ

```
TOOL a, b, c;
(*bin)[0] = a;
(*bin)[1] = b;
(*bin)[2] = c;
```

ส่วนการเข้าถึงข้อมูลข้างใน TOOL ทำได้โดย

```
int i;
/* กำหนดตัวแปร integer */
i = (*bin)[0].field;
/* วิธีการเข้าถึงข้อมูล */
```

ตอนนี้มาลองแบบที่เป็นฟังก์ชันบ้าง

```
typedef struct {
/* กำหนดชนิดของ TOOL */
int field;
} TOOL;
TOOL (*bin) ( );
```

เรารู้ว่า bin คือตัวชี้ไปยังฟังก์ชันที่คืนค่าเป็น TOOL ต่อไปก็ประกาศ a, b, และ d โดย

```
TOOL a( );
/* a เป็นฟังก์ชันที่ return TOOL */
TOOL b;
/* b เป็นข้อมูลชนิด TOOL */
int d;
/* d เป็นข้อมูล integer */
```

ขั้นตอนคือใส่ address ของฟังก์ชันลงไปใน bin โดย bin=&a แต่ความจริงแล้ว ชื่อของฟังก์ชันที่ไม่มีวงเล็บต่อท้ายจะหมายถึงตำแหน่ง (address) ของฟังก์ชันนั้นแล้วโดยไม่ต้องมีสัญลักษณ์ & คล้าย ๆ กับการเรียกชื่อของอะเรย์โดยไม่มี subscript ตามหลังจะหมายถึงตำแหน่ง (address) ของต้นอะเรย์ ดังนั้นการเข้าถึงข้อมูลข้างในทำโดย

```
bin = a;
/* กำหนดค่าของตัวชี้ (bin) */
b = (*bin)( );
/* เรียกใช้ฟังก์ชัน */
d = b.field;
/* การเข้าถึงข้อมูลข้างใน */
```

ประโยชน์

หลักเกณฑ์ที่กล่าวมานั้นใช้ได้กับการกำหนดตัวแปร (declaraction), typedef, และ typecast ตัวอย่างที่ยกขึ้นมาจะเห็นว่าจากง่ายไปหายาก

สำหรับที่มีใช้อยู่จริง ๆ เช่นใน include file มาตรฐานของ UNIX ชื่อ /usr/include/sys/signal.h เราจะพบ

```
#define __NSIG 33
#define SIG__IGN ((void (*) ( ))
(__NSIG + 10)
```

เราจะถอดเจ้า SIG__IGN นี้ออก เนื่องจากว่าเป็น typecast และถ้ามีตัวแปรจะอยู่ตรงหลังสัญลักษณ์ * ดังนั้น typecase นี้จะอ่านว่า "pointer to function returning void" หรือ "ตัวชี้ไปยังฟังก์ชันที่คืนค่าเป็น void หรือไม่คืนค่านั่นเอง" ตัวอย่าง typecast ง่าย ๆ คือ (char *) 0 กำหนดให้ตัวชี้ไปยัง char มีค่าเป็น 0 ดังนั้น SIG__IGN นี้ก็จะให้ตัวชี้ ชี้ไปยังหน่วยความจำบริเวณที่เป็นจุดเริ่มต้นของฟังก์ชันหนึ่ง ซึ่งเป็นตัวอย่างของงานโปรแกรมระบบ (system programming) ที่รู้ตำแหน่ง (address) ของฟังก์ชันภายใน ลองดูตัวอย่าง

```
TOOL *(*(*bin)( )) ( );
```

ทดลองว่าอ่านได้แบบนี้หรือเปล่า "bin is a pointer to a function returning a pointer to a function returning a pointer to TOOL" หรือ "bin เป็นตัวชี้ไปยังฟังก์ชันที่จะคืนค่าเป็นตัวชี้ไปยังฟังก์ชันที่ให้ค่าเป็นตัวชี้ TOOL"

สุดท้ายนี้ขอยกตัวอย่างโปรแกรมในรูปที่ 6 ที่ดูเหมือนเขียนแบบเข้ารหัสในตัวเลยก็ว่าได้ แทนที่จะเขียนให้อ่านง่าย ๆ เป้าหมายก็แค่เรียกฟังก์ชันเพื่อให้ได้ค่า 123 กลับมาเอง การอ่าน-เขียน-ใช้ C declaration

```
typedef struct {                        /* กำหนด structure TOOL */
        int field;
}TOOL;
typedef TOOL                            *(*(*BINA__DEF) (void)) (void);
typedef TOOL                            *(*A__DEF) (void);
typedef TOOL                            *(*BINB__DEF) (void);
typedef TOOL                            *B__DEF;
typedef TOOL                            *C__DEF;
/* เก็บตัวชี้ไปยังฟังก์ชันที่คืนค่าเป็นตัวชี้ไปยังฟังก์ชันที่คืนค่าเป็นตัวชี้ไปยัง TOOL */ BINA__DEF bina;
A__DEF      a(void);                    /* function prototype ของฟังก์ชัน a */
/* เก็บตัวชี้ไปยังฟังก์ชันที่คืนค่าเป็นตัวชี้ไปยัง TOOL */
BINB__DEF binb;
B__DEF      b(void);                    /* function prototype ของฟังก์ชัน b */
C__DEF      c;                          /* เก็บตัวชี้ไปยัง TOOL */
TOOL        tool;                       /* ตัวแปรที่เป็น structure */
int         d;                          /* ตัวแปร integer ซึ่งชนิดเดียวกับ field ใน TOOL */
main( )
{
        bina = a;
        binb = (*bina) ( );             /* ได้ค่าคืนมาเป็นตัวชี้ไปยังฟังก์ชัน */
        c = (*binb) ( );                /* ได้ค่าคืนมาเป็นตัวชี้ไปยัง TOOL */
        d = c->field;                   /* การ access ข้อมูลใน structure TOOL */
        printf("d (should be 123) has value %d \n", d);
        return(0);
}
A__DEF      a( )
{
        return(b);                      /* คืนค่าเป็นตำแหน่งของฟังก์ชัน b */
}
B__DEF      b( )
{
        tool.field = 123;
        return(&tool);                  /* คืนค่าเป็นตำแหน่งของ structure */
}
```

รูปที่ 6 ตัวอย่างโปรแกรมที่แสดงการใช้ declaration โดยต้องการเพียงแค่ส่งค่า 123 มาให้ฟังก์ชัน main( ) แสดงค่าเท่านั้น